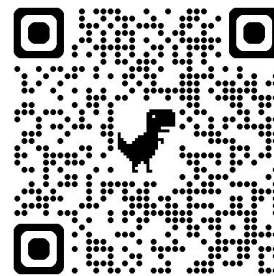


The era of generative content creation using artificial intelligence

Era generatywnego tworzenia treści z wykorzystaniem sztucznej inteligencji

— Piotr Chlebek

<https://www.linkedin.com/in/piotrr/>



Before we start...

Credits to

- Historical images (inc. this background) <https://wikimedia.org/>
- Contextual illustrations: Gemini 2.5 Flash Image (Nano Banana)
- More sources inline

Opinions and views expressed in this presentation are solely my own or quoted. They are not related to any company for which I work / worked.

In Polish: Opinie i poglądy wyrażone w tej prezentacji są wyłącznie moje własne lub cytowane.

Nie są powiązane z żadną firmą, dla której pracuję / pracowałem.

Feedback or questions ?

I encourage You to give me feedback or ask a questions. Yes, You can interrupt me during the presentation.



About me



Piotr Chlebek

- ~25 years of experience, R&D a wide range of innovative projects

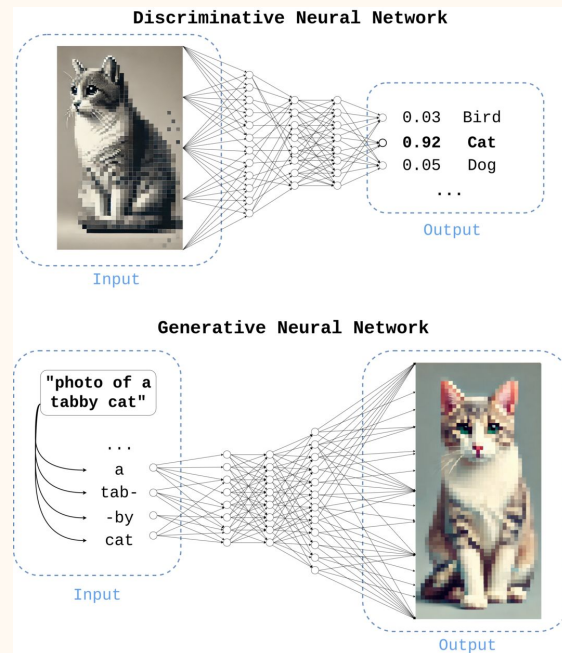
Speech Recognition & Machine Learning projects, successful products on the market with industry recognition & awards.

- ML/DS passionate
 - [ML Gdańsk](#) community active member
 - [Sharky Neural Network](#) - software for education
 - Chess programming
- I Love Delphi (hobby++)

What is GenAI?

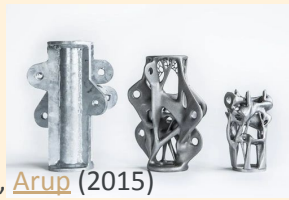
Generative artificial intelligence (Generative AI, GenAI, or GAI) is a subfield of artificial intelligence that uses generative models to produce text, images, videos, audio, software code or other forms of data. These models learn the underlying patterns and structures of their training data and use them to produce new data based on the input, which often comes in the form of natural language prompts.

Source: https://en.wikipedia.org/wiki/Generative_artificial_intelligence

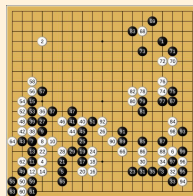


A short history of Gen AI

Sources: <https://www.igmguru.com/blog/history-of-generative-ai>, <https://en.wikipedia.org/>, Arup (2015)



1945 - ENIAC
1996 - 3dfx Voodoo
2001 - IBM Power4
2002 - Sony PlayStation 2
2006 - Intel Core Duo
2007 - Nvidia CUDA
2016 - Google TPU



Source Code Gen AI
2021 - OpenAI Codex (GPT-3 based) trained on public GitHub code
2021 - GitHub Copilot - the first generative AI coding assistant
2023.q4 - "Agentic Coding"
2025.02 - "Vibe Coding"
2025.05 - "Spec-Driven AI Development"

1950+

1980

1996

2014

2016

2017

2021

2023+

Birth of AI

Artificial Intelligence introduced as a scientific discipline at the Dartmouth Conference. Perceptron (1958). ELIZA (1961).

The Rise of NN

Recurrent Neural Networks (RNNs; 1982) & Long Short-Term Memory networks (LSTMs; 1997) for sequences. Variational Autoencoders (VAE; 2013)

GANs & Diffusion Models

Generative Adversarial Networks (GANs; 2014) have revolutionized image generation with adversarial training. Diffusion Models (2015).

Recent Advances in Gen AI

Transformer Architecture (2017). Generative Pre-trained Transformer (GPT; 2018) breakthrough in natural language generation using transformers.

The Rise of text-to-image

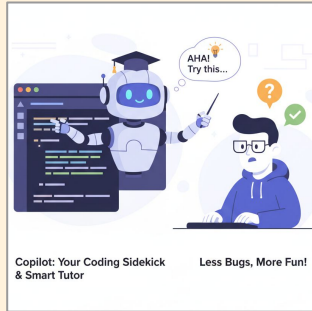
DALL-E (2021), Rise of high-quality, text-to-image tools including open-source Stable Diffusion and artistic Midjourney (2022).

"Blooming Meadow"

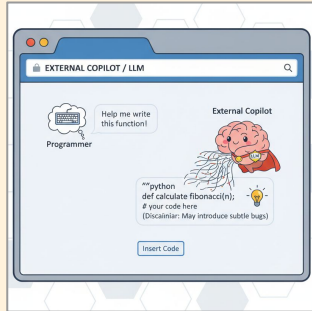
Multimodal & long context - GPT-4 (2023). Emergence of text-to-video - Sora, Claude 3, Gemini 1.5 (2024). AI Code Agents & On-Device AI (2025).

How to use GenAI

Access methods



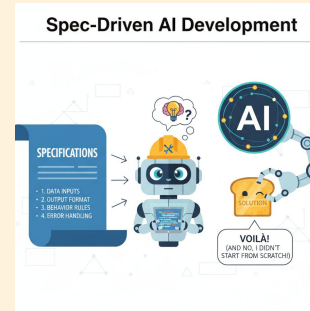
Integrated Copilot,
IDE help & tutor



External Copilot / LLM
or External Tools



Vibe Coding

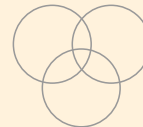
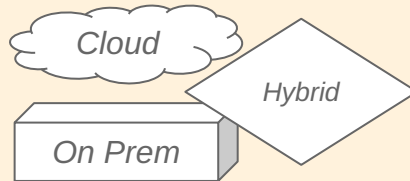


Spec-Driven AI
Development



Agentic Coding

Yes, you can
integrate external
LLMs inside!



Yes, it can overlap!

What can GenAI generate?

[S] Small: code completion, code snippets, comments & docstrings, summaries, functions, procedures, interfaces, functional tests, unit tests, property/method stubs, wrappers, SQL snippets, Regex, commit messages, changelog entries, code review, error explanations, debug hints, the resources, localized texts, ...

[L] Large: classes, units, forms, components code, REST API clients/servers, refactor patches, serialization, database layers, data model design, test suites, documentation, collection of resources, ...

[XL] Extra Large: new project templates, sample apps, entire services, middleware layer, entire apps, entire systems, agentic workflows, automation bots, large refactors, larger documentation, ...

[!] But: GenAI often gets it wrong, causing mess and added complexity.



Practical applications in the broad context of Delphi

Brainstorm

GenAI (potentially) in the Delphi IDE environment

Runtime

Autocomplete,
Auto-(re)implement,
Code Conversion,
Suggestions,
Adding Comments,
Decomposition,
3rd party Integration,
Localization, ...

Completion,
Code/UI generation,
Refactoring,
Code modernization,
Code translation

Summarize, Analyze,
Estimate, Explain,
Generate: Resources,
Test Cases, Docstrings,
Specs/UML, Mocks/Stubs,
Changelog, Release Note,
Help, ...

Explanation,
Documentation,
Test,
QA

AI-assisted component design,
GenAI-driven debugging,
Conversational compiler errors,
AI pair programming for Delphi,
FireDAC + LLM - data querying
and natural summaries,
...

Experimental

Embedded:
- Models (e.g. LLM)
- Chatbots
- Voice assistants
- Natural-language
commands
...

Embed, Integrate

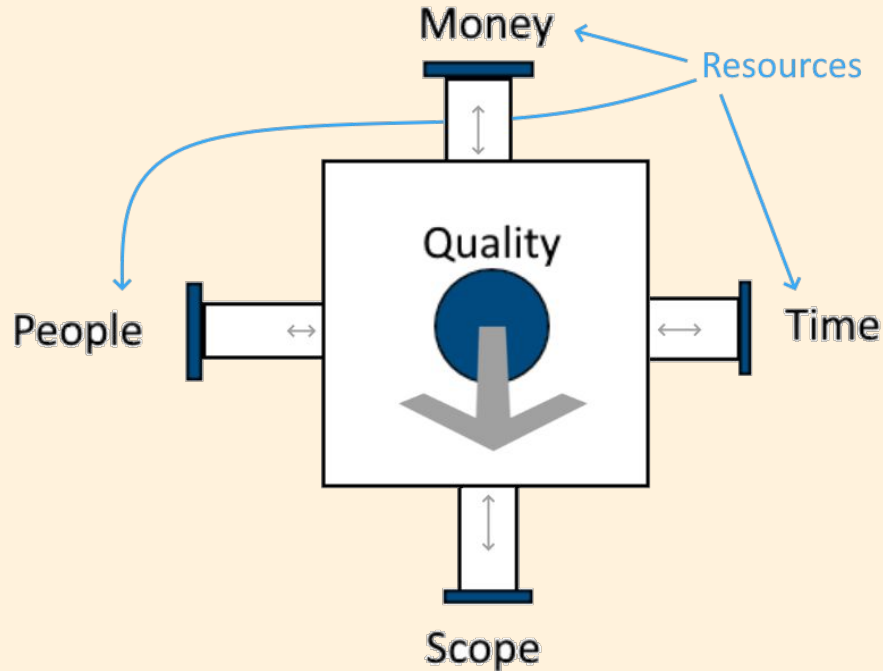


Wait! We're developers, but what about
creating GenAI applications in Delphi?

Impact

Machine of five states perspective (1/4)

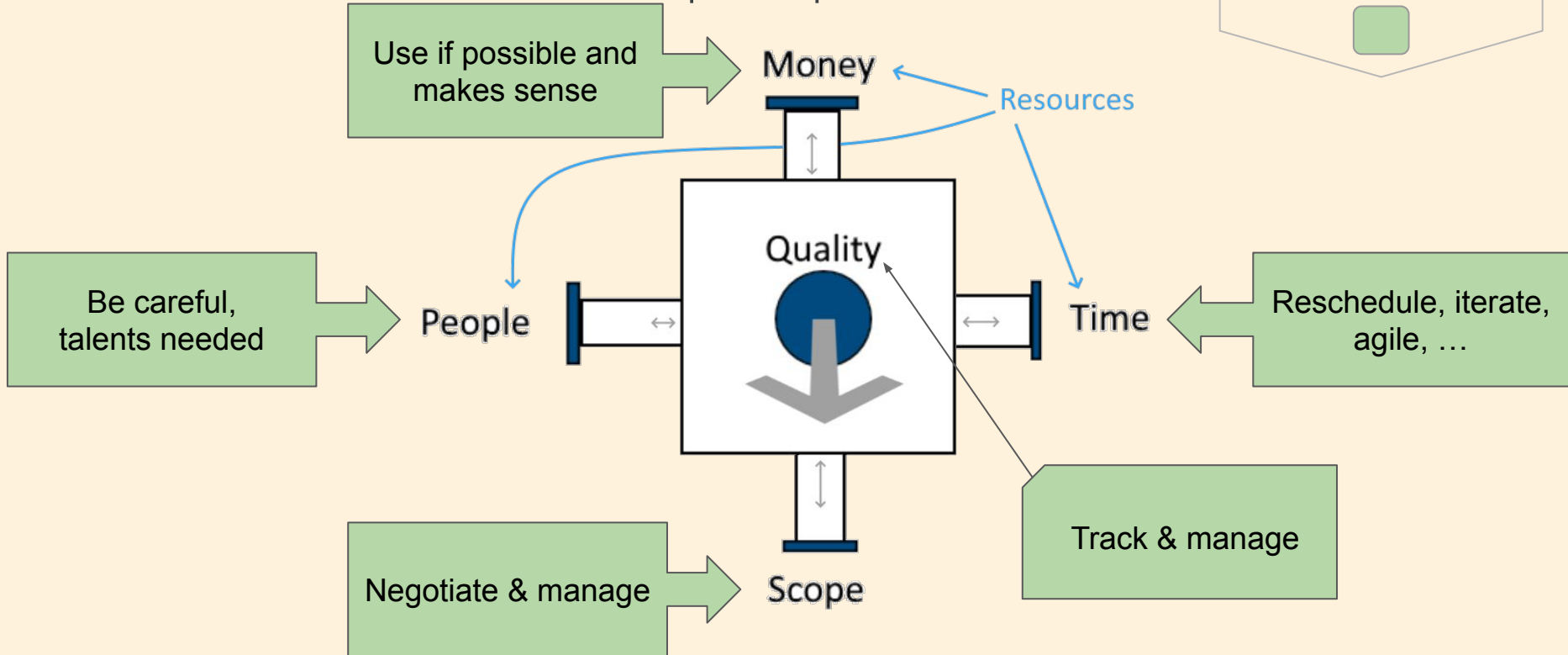
Idea source: High risk projects by dr **Tadeusz Lis**



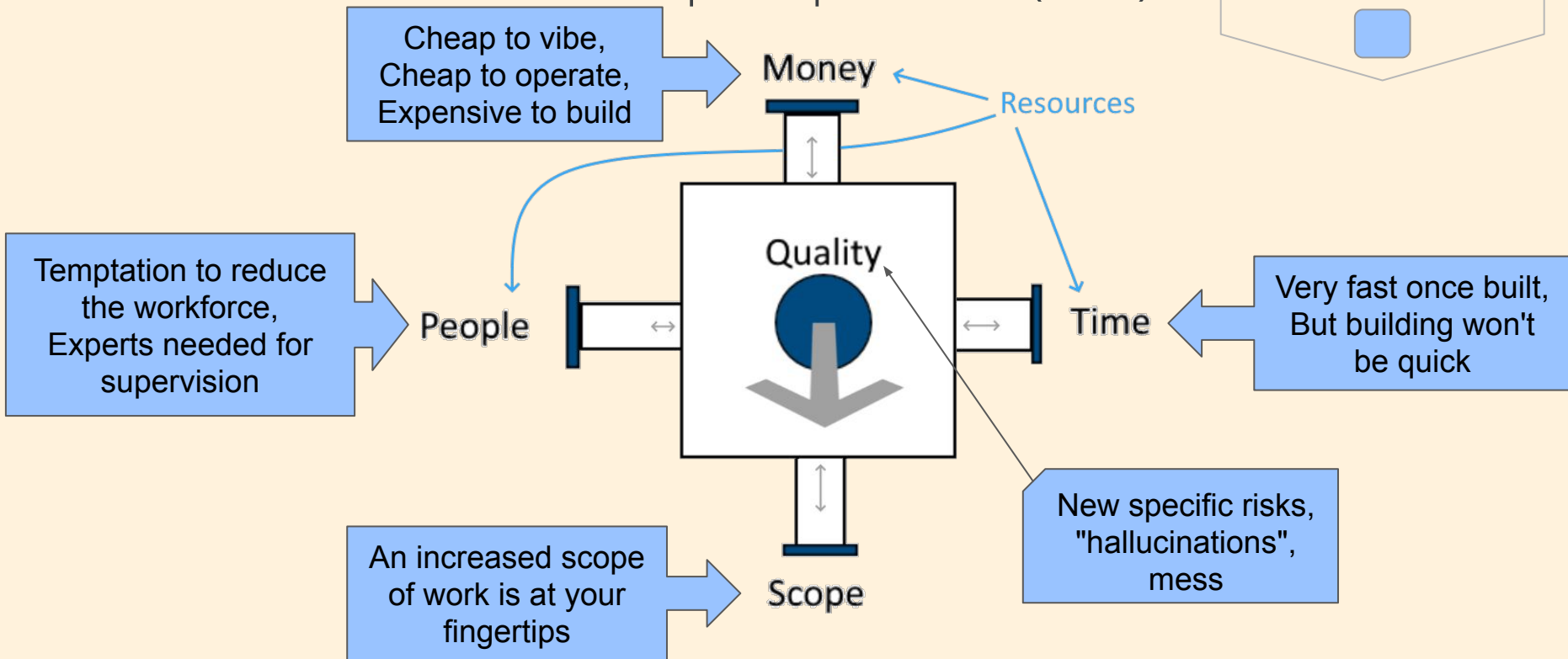
Impact

Machine of five states perspective (2/4)

How to deal with it



Machine of five states perspective (3/4)



Impact

Machine of five states perspective (4/4)

GenAI
perspective



Cheap to vibe

C

But that's not all!

Time-to-Market++, Faster Iterations & feedback,
Rapid Prototyping / POC / MVP,
Higher Productivity, Better Focus (Less Cognitive
Load), Scalability++, Cross-Platform Support++, API
Integration++, Interoperability++, Legacy
Modernization++, Test Coverage++, Architecture
Compliance++, UX/UI Ideation++, Design
Exploration++, Domain Intelligence++, Onboarding++,
Knowledge Retention++, Communication++, ...

:(New kinds of problems/bugs are blooming!

:(Causing mess and adding complexity!

Temptation to reduce
the workforce,
Experts needed for
supervision

Very fast once built,
But building won't
be quick

risks,
ons",
s

fingertips

Let's compare the approaches



Approach

Vibe Coding

Conversational

Developer Role

Prompt engineer

AI Interaction

Iterative refinement

Best Suited For

Rapid prototyping, solo dev

Scalability

Very Low

Maintainability

Low

Upfront Effort

Minimal

Flexibility

High

Observability

Prompt logs and diffs



Spec-Driven AI Development

Structural

Specification author

Guided by specifications

Structured projects

Moderate*

High

High

Moderate

High



Agentic Coding

Autonomous, goal-oriented

Goal setter

Autonomous execution

Large-scale applications

High

Moderate

High

Low*

Low

GenAI examples

Vibe Coding - Almost any SOTA LLM.

Spec-Driven AI Development

1. **/speckit.constitution** - governing principles
2. **/speckit.specify** - what & why (not tech stack)
3. **/speckit.plan** - tech stack & architecture
4. **/speckit.tasks** - actionable task lists
5. **/speckit.clarify** - underspecified areas
6. **/speckit.analyze** - analyze existing source code
7. **/speckit.implement** - execute all tasks
8. **/speckit.checklist** - quality checklists

1. (Sep. 2025) Github Spec Kit (MIT) - <https://github.com/github/spec-kit>
2. (Aug. 2025) SDD Flow (MIT) - https://github.com/Ataden/SDD_Flow
3. Better Spec - <https://better-spec.com/>
4. SpecifyX - <https://www.specifyx.dev/>
5. SpeckJS - <https://speckjs.github.io/>
6. Spec-Driven - <https://mcpmarket.com/server/spec-driven>

Agentic Coding - Read: [Top Frameworks for Building Agentic AI Systems in 2025](#)

(6th Oct. 2025) OpenAI AgentKit - <https://openai.com/index/introducing-agentkit/>

GenAI examples

Delphi perspective

Vibe Coding - Almost any SOTA LLM.

Other:

MakerAi (MIT) - <https://github.com/gustavoeenriquez/MakerAi>

Delphi-AI-Developer (MIT) - <https://github.com/Code4Delphi/Delphi-AI-Developer>

Delphi GenAI (MIT) - <https://github.com/MaxiDonkey/DelphiGenAI>

Delphi StabilityAI API (MIT) - <https://github.com/MaxiDonkey/DelphiStabilityAI>

DelphiOpenAI (MIT) - <https://github.com/Hemu1GM/DelphiOpenAI>

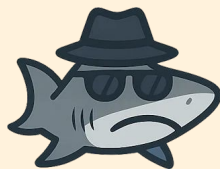
OpenAI-Delphi (CC-0) - <https://github.com/magnolima/OpenAI-Delphi>

OpenAI for Delphi and Lazarus/FPC (MIT) - [landgraf-dev/openai-delphi](https://github.com/landgraf-dev/openai-delphi)

Lumina (BSD-3) - <https://github.com/tinyBigGAMES/Lumina>

Delphi Parser - <https://delhiparser.com/>

Online Delphi Code Generator - <https://www.codeconvert.ai/delphi-code-generator>



or THTTPClient,
TNetHTTPClient



GenAI specification content examples

1. Constitution - what must always be met
2. Vision (what & how)
3. Technology stack; compiler, frameworks, dependencies, libs, APIs, ...
4. List of tools and their capabilities
5. High/low level Architecture, existing code
6. Requirements, User Stories, Tasks
7. Coding standards, industry/domain standards, methodology, processes
8. Risk plan, Test plan, Quality plan, Integration plan, Deployment/Release plan
9. Data & Domain specifications
10. UI / UX specifications
11. Documentation & Metadata specs

GenAI applications in Delphi

Why?

	Web Page	Delphi Desktop Apps
Rich client / Ecosystem	Yes / Awesome	Yes / OK
Images / Text / Audio Video	OK / OK / Moderate	OK / OK / OK
Spell check	OK	OK
Responsiveness	Moderate	Awsome
Upgradability	Awsome	OK
Privacy	Moderate	OK
Local documents	Low	Awsome
Cloud documents	Awsome	Low
Team work	Awsome	Demanding

Best practices

Delphi developer perspective

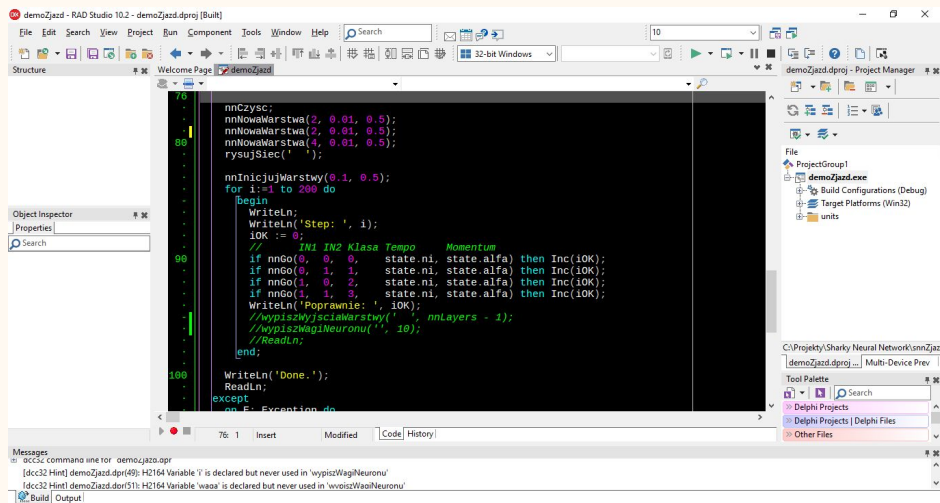
- Data, data, data! - transparency, observability, metrics, evaluation, ...
- See and think about systems & structures (structured & versioned specification)
- Testing AI systems is difficult, automate (run/create), enable human-in-the-loop
 - Design for scenario when something goes wrong
 - Validate AI outputs automatically, use guardrails
- Layered architecture (spec parsing, AI logic, UI, data, prompts / templates, GenAI Wrappers, ...)
- Prompts & specs:
 - Separate prompts & specs from code, version & log them
 - Use content templates (with placeholders) for prompts and specs
 - Cache prompts & results, retry, async vs. sync, separate/hide keys

Best practices

Embarcadero perspective

- Delphi integrations
 - Built-in vs. optional
 - Delphi as an agent tool, and other tools (e.g .static code analysis)
 - Deployable blocks (e.g. embedded LLMs)
- Enabling model improvement
 - Data, data, data! (friendly license)
 - Code, articles, documentation, tutorials, blogs, discussions, ...
- Legacy nightmare - version & dependency attribution, code translations
- Community

Code Demo



Questions?
Discussion
